The Acoustic Analysis Workbench

Keith Bromley, Bob Dukelow, Jerry Symanski
SPAWAR Systems Center, San Diego
aaw@spawar.navy.mil

Abstract

This paper describes a set of software components, called the Acoustic Analysis Workbench (AAW), that can be used for a broad set of applications in the real-time analysis and visualization of acoustic input signals. As a simple example, some of these components can be interconnected to calculate and display a sliding-window spectrogram (i.e., a short-time Fourier transform) of a single-channel input time-series. A more complex arrangement of these components can perform beamforming from an array of acoustic sensors (hydrophones) and compute the spectrogram of every beam. Still other components can be added to this arrangement to transpose this output data into a time vs. beam-number display for each frequency, etc. In general, end users of this application will be naval scientists and engineers exploring algorithms for signal transformation and feature extraction from acoustic time-series (typically from arrays of sonar sensors).

Underlying Foundations

The basic premise of this work rests on three underlying foundations: (a) rapid advances in the capabilities of modern computers, (b) the ubiquity of the Microsoft Windows OS, and (c) the recent availability of good programming tools for streaming data-flow processing.

High-performance computing has historically been thought of as most appropriate for "grand challenge" scientific computations (e.g., computational fluid dynamics and structural mechanics) - but generally not appropriate for real-time or embedded applications. However, the relentless continuation of Moore's Law in improving microprocessor performance and the increasing commercial deployment of parallel computer architectures are rapidly providing systems which are extremely capable for real-time signal/image processing.

For the Intel X86 architecture, indications of the dramatic growth in computational capability are the increase in transistors-per-die from a lowly 29K for the 8086 chip in 1978 to 42M for the Pentium 4 chip today, and the impressive increase in clock frequency from 8 MHz then to 1.7 GHz today. Several manufacturers have announced parallel supercomputers employing up-to-32 of the latest Intel chip - the Itanium.

Microsoft Windows is the de-facto dominating operating system (OS) in both the personal computing and commercial/business computing worlds. It is used in over 90% of all computers, and almost all current high-school graduates in the U.S. have some familiarity with using it. It is clear that there is significant potential gain to writing applications software such that the user-interface is in a graphical environment familiar to most users. Microsoft produces a version of their OS called the "Windows 2000 Datacenter Server" which works on up-to-32 shared-memory parallel processors.

Microsoft provides middleware (a hierarchical set of C++ objects, associated run-time libraries, and tools) called DirectX that simplifies the programming of signal/image processing applications. DirectX's run-time libraries are automatically loaded during standard Windows installation; the application programming interfaces (APIs) are well defined and documented; and it's software development kit (SDK) is freely available from the Microsoft web site.

While Microsoft's motivation for developing the DirectX APIs is to further their penetration into the computer gaming and audio/video streaming markets, it is the conjecture of the authors that this technology can be fruitfully applied to serious sensor signal processing. Furthermore, to flourish in these two mass-market application areas, Microsoft is highly motivated to improve their DirectX implementations as Intel improves their microprocessors by adding hardware support for new microinstructions for multimedia extensions (MMX) and streaming-SIMD extensions (SSE). Hence, applications written to the high-level DirectX API will automatically improve their run-time performance by making very efficient use of the latest Intel architectural enhancements and instruction-set tweaks - all invisibly to the application developer or end user.

The key DirectX component for this project is DirectShow, which implements a data-flow filter-graph programming paradigm in which data is streamed through a graph of filters. There are three types of filters: source, transform, and renderer. Source filters retrieve data from a media source (such as a microphone, a camera, a local data file, or the internet). Transform filters manipulate data and pass it to the next filter in the graph. The arrangement of transform filters defines the processing algorithm. Finally, renderer filters output the data to some external transducer - such as a loudspeaker, a video monitor, a data file, or the internet.

The Acoustic Analysis Workbench

The Acoustic Analysis Workbench (AAW) is a set of software components (i.e., DirectShow filters) that can be used for a broad set of applications in the real-time analysis and visualization of acoustic input signals. It is a highly flexible “problem-solving environment” for audio decomposition. As a simple example, some of these filters can be interconnected to calculate and display a sliding-window spectrogram (i.e., a short-time Fourier transform) of a single-channel input time-series. A more complex arrangement of these components can perform beamforming from an array of acoustic sensors (microphones) and compute the spectrogram of every beam. Still other components can be added to this arrangement to transpose this output data into a time vs. beam-number display for each frequency, etc. The user typically uses a Microsoft application called GraphEdit (available in the DirectX SDK) as a graphical point-and-click "programming environment" for selecting the filters, arranging them into a complete graph, and running the resultant graph with the selected input data - which may be a file or a "live" streaming input (e.g., a microphone or a sensor array).

The AAW software is written in C++ and, at the completion of this project, all source code will be openly released into the public domain. (Although Microsoft only distributes binaries for its DirectShow runtime system and it’s graph editing application.) The most computationally demanding filters have been written to take advantage of multithreading. Hence they will achieve scalable speed-up as they are moved from a

single-CPU desktop PC to a multi-CPU workstation to a symmetric-multiprocessing supercomputer. For computational efficiency, the code makes use of the Intel Signal Processing Library (SPL) - a set of machine-language implementations of common signal processing functions. Thus some of the AAW code will only run on Intel CPUs.

AAW FILTERS

This section presents a brief overview of the individual AAW filters. (These are generally in the order in which they would typically appear in a graph - except for the final two which are utility functions which can go almost anywhere.)

AAW File Source (AAWFileSource.ax)
The AAW File Source filter is a source filter designed to read time-domain data from a .wav file and feed it to downstream filters. Note that the sample rate of the output data stream can be different than the sample rate read from the data file - which is useful if either the data rate on the file is incorrect or if it is desired to render the data at a rate which is faster or slower than real time.

AAW Frequency Domain Beamformer (AAWFDBF.ax)
The AAW Frequency Domain Beamformer processes an input stream that consists of multiple channels of time-domain data samples. It is assumed that each input channel represents the time series from one hydrophone element of a sonar array. The output stream from this filter is some number of channels of time-domain data samples representing beams formed in specific physical directions. The number of output channels (beams) is independent of the number of input channels (elements). The algorithm currently assumes isovelocity propagation of the sound field and the beamforming is fixed (i.e., there is no adaptive processing).

The basic concept is that a short-time Fourier transform is applied to each of the data channels to produce an array of narrow-band time samples (at a reduced sample rate) for each channel. For each frequency bin the input-channel (element) data are combined with appropriate phase delay (and array amplitude shading if desired). For each beam, all of the frequency bins are then recombined into a single time-series output stream at the original sample rate using an inverse Fourier transform.

The current implementation assumes that the array is a straight line, that the channels are simultaneously sampled, and that the elements are linearly spaced along the array at a user-selected spacing. Since the actual beamforming is performed using a phase weight table that is computed during run initialization, the code could easily be modified to accommodate other array configurations, beam steering directions, and/or data sampling schemes. The current implementation was limited primarily to avoid to complexity of having the user input N arbitrary X, Y, Z positions for the hydrophone elements or individual beam-steering angles.

AAW FFT (Fast Fourier Transform) (AAWFFT.ax)
The AAW FFT filter performs a power-of-2 length Discrete Fourier Transform (DFT) operation on each of an arbitrary number of channels of time-domain, real-valued input data. The output contains the same number of channels as the input but each output time sample for each channel is a complex frequency vector

of non-negative frequency bins. (The number of complex frequency bins is half the input block size.) The user may select (using the property page) the DFT block length, the block overlap in percent, and a shading window function to apply to each input data block prior to the DFT.

AAW Constant-Q Analysis (AAWConstQ.ax)

AAWConstQ performs constant-Q spectral analysis on a real-valued, time-domain input data stream. The term constant-Q implies that the analysis bandwidth of each frequency bin is proportional to it's center frequency. Conceptually, the spectral analysis is accomplished with a bank of bandpass filters whose center frequencies are linearly spaced in log frequency. The ratio of center frequency over 3 dB bandwidth is the same for all band filters (i.e., all the filters have the same Q) and the upper 3 dB cutoff frequency of one band filter is the same as the lower 3 dB cutoff frequency of next higher band filter. The input stream may contain an arbitrary number of channels that are each analyzed independently. The output contains the same number of channels as the input but each output time sample for each channel is a complex frequency vector.

The basic analysis approach and filter algorithms are adapted from [1]. The bandpass filter center frequencies and bandwidths are designed such that a user-selectable integer number of bands exactly spans (to the 3 dB points) an octave. First, a set of band filters is designed to analyze a single octave. After analyzing the first octave the data is lowpass filtered and decimated by two so that the same set of filters (running at the reduced sample rate) can be used to analyze the next lower octave. The process of lowpass filtering, decimating, and analyzing an octave is repeated for the number of octaves specified by the user.

AAW Decimatation Filter (AAWDecimate.ax)

The AAW Decimate filter provides both temporal decimation (i.e., time resampling) and bin windowing (i.e., the output bins can be a contiguous subset of the input bins). Note that "bins" often correspond to frequencies but this may not always be the case. Only complex valued, floating-point input and output data types are currently supported. This means that this filter must generally be placed downstream from an AAWFFT or AAWConstQ filter and prior to the normalizer. There may be other types of filters before or after AAWDecimate (e.g., AAWChanSelect, AAWChanTee, and AAWTranspose) within this sequence. The number of output channels is always the same as the number of input channels. Channels can be reduced or rearranged using AAWChanTee or AAWChanSelect.

For temporal decimation, the user selects a "Temporal Decimation Ratio" (denoted by "*R*") which is an integer greater than or equal to 1. The output sample rate is equal to the input sample rate divided by *R*. The user can currently select one of two decimation schemes:

1 - "discrete resampling" simply outputs one out of every *R* input samples.

2 - "maximum over range" outputs the maximum magnitude sample out of each successive group of R input samples.

For bin windowing, the user selects a starting and ending input bin number, where the first bin is 0. Only input bins that fall between the start and end bins (inclusive) are copied to the output stream.

AAW Normalizer (AAWNorm.ax)

The AAW Normalizer filter implements several normalization methods converting complex floating-point vector data into 8-bit unsigned integers which can be directly mapped to color/intensity values for display by

the renderer filter. The input data stream may contain any number of channels where a "time sample" for each channel is a vector of complex bins. A bin most commonly represents a frequency. All channels presented to the normalization filter are independently processed and output together in a single output data stream.

Using the property page, the user can select one of eight normalization schemes. These include maximum-over-line (signal magnitude), maximum-over-line (signal power), fixed gain (signal magnitude), fixed gain (logarithmic output),) Two-Pass Mean, Order Truncate Average, Split Three-Pass Mean, and Split Average Exclude Average. The last four are described in [2] and use various statistical techniques to better visualize weak signals buried in noise.

Unlike most of the other AAW filters, parameter changes made while this filter is running take effect immediately for the next full data block processed. This allows the user to interactively test the effect of various normalization schemes and parameters such as gain and/or dynamic range.

AAW Transpose (AAWTranspose.ax)
This filter takes in a stream of data where the dimensions of the data are referred to as time, channel, and bin. Typically channels will be different hydrophone elements or beams and bins will be a frequency decomposition of the data. AAW Transpose transposes the channel and bin dimensions so that, for example, a renderer that displays time versus bin for a given channel can be used (unchanged) to display time versus channel for a given bin.

AAW Renderer SC (Single Channel Renderer) (AAWRendererSC.ax)
The Single Channel Renderer filter displays the results of signal processing by upstream AAW filters. The input pin of the Single Channel Renderer is always downstream from an AAW Normalizer filter. The data comes to the Renderer as a buffer of bytes. Since the user may wish to view earlier data and move around in a large data buffer, the renderer window has scrollbars so the user can view all of the data. Other useful features are; the ability to change the color map, data magnification, and indication of data values under the cursor. There are four display types for the Single Channel Renderer:

Top-to-bottom - old data moves down,
Bottom-to-top - old data moves up,
Left-to-right - old data moves horizontally to the right, and
Right-to-left - old data moves from right to left.

The data is received into a large fixed-size circular buffer memory and stored as bytes. The bytes will be displayed using the user-selected color table in a window appropriate to the available screen size and color depth. For instance, a 1024 point FFT would result in 512 frequency bins across the width of a top-to-bottom renderer window. The user is able to select the size of the window and the (horizontal and vertical) magnification of the data. At a minimum, each data element is represented by a single screen pixel.

Figure 1 shows a sample output of the AAW Renderer displaying a spectrogram of some whale sounds with time progressing from top to bottom and frequency increasing from left to right. The static picture shown in Figure 1 does not do justice to the fact that the user actually sees a dynamic scrolling "waterfall" display with the instantaneous spectrum at the top of the screen in real-time synchronization with the sound being processed and (if desired) played through the computer's speakers.

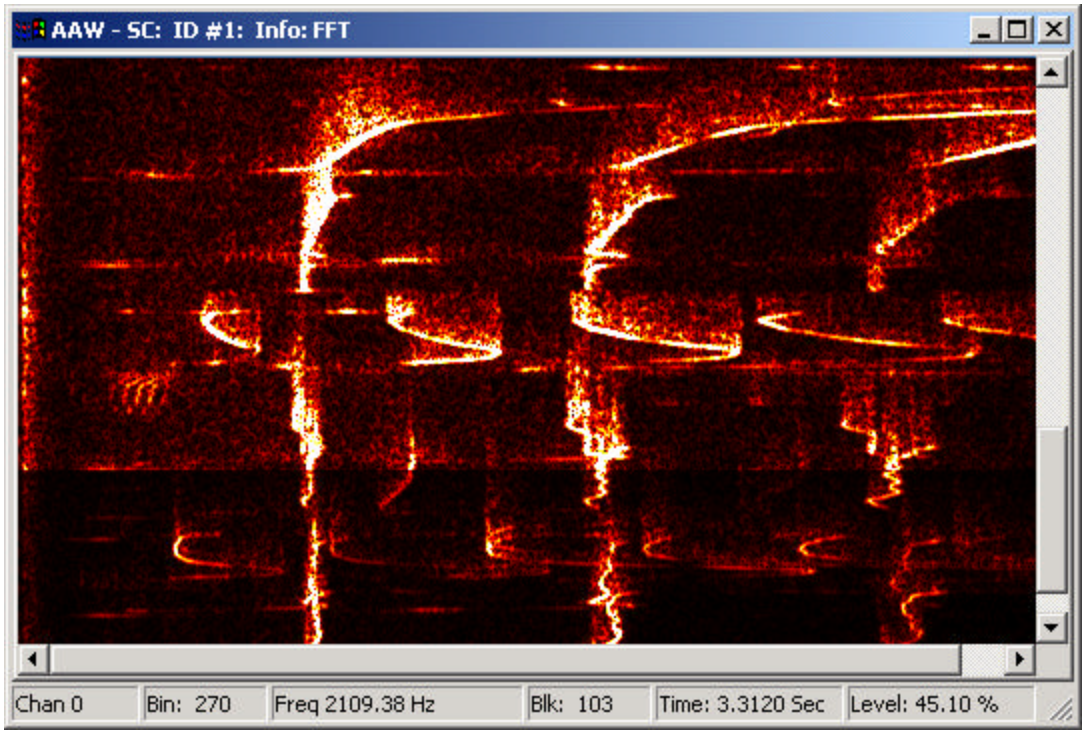

Fig. 1. *Sample Output of the AAW Renderer. Time progresses from top to bottom and frequency increases from left to right.*

AAW Renderer MC (Multi Channel Renderer) (AAWRendererMC.ax)

The multi channel renderer differs from the single channel renderer in that multiple channels are displayed in one window. The filter determines how many channels are in the incoming data and calculates how many window “panes” to use. The user can specify how many channels to display, the first channel to show, and the number of rows and columns to use in positioning the window “panes”.

AAW Channel Tee (AAWChanTee.ax)

AAWChanTee converts a single data stream (input pin) containing multiple channels to one or more data streams (output pins) each containing only a single channel. The user may select which input channel is mapped to each output pin. Different output pins may contain data from the same or different input channels. The filter starts with a single output pin, but output pins are added as output connections are made so that there is always a spare output pin to allow another connection.

AAW Channel Select (AAWChanSelect.ax)

The AAWChanSelect filter converts a single data stream (input pins) containing multiple channels to one or more data streams (output pins) containing (potentially) different channels than the input stream. All output pins contain the same data stream but this data stream is a user-defined subset of the input-stream channels. The filter starts with a single output pin, but output pins are added as output connections are made so that there is always a spare output pin to allow another connection. The user can set both the number of output channels (which may be more or less than the number of input channels) and the mapping between input and output channels.

Scalability

While our work in the coming year will emphasize scalability improvements, some preliminary experiments have been performed. Simulated data from a hypothetical 128-element sonar array, with a data sample rate of 1000 Hz, was processed through a graph containing the File Source, Frequency-Domain Beamformer, Channel Tee, Normalizer, and single-channel Renderer filters. 128 beams were computed and one was selected for subsequent analysis and sliding spectrogram display using a 2,048-point DFT with a 50% overlap. The measured performance was

CPU's	Execution Time	Measured Speedup	Ideal Speedup
1	248.4 sec	1.00	1.00
2	135.0 sec	1.84	2.00
4	79.5 sec	3.12	4.00
8	63.5 sec	3.91	8.00

Conclusions

The authors hope that this software will stimulate a community of signal-processing programmers to further add to it's usefulness. We invite others to fruitfully expand upon it's capabilities by developing more DirectShow filters for interesting audio transformations.

Acknowledgements

This software was developed under the SIP-6 project within the Signal/Image Processing computational technology area within the Common HPC Software Support Initiative (CHSSI) under the DoD High Performance Computing Modernization Program. We wish to thank Dr. Richard Linderman, Lt. Col. Jim Albert, Mr. John Grosh, and Dr. Leslie Perkins for their continued support of this effort.

References

1. F. Harris, “An Efficient Constant-Q Analyzer Architecture Using All-Pass Recursive Filters”, San Diego State University, (to be published).

2. W. Struzinski and E. Lowe, “A Performance Comparison of Four Noise Background Normalization Schemes Proposed for Signal Detection Systems”, *J. Acoust. Soc. Amer.*, Vol. 76 (6), 1738-1742, December 1984.